特別研究報告

題目

SZZ 手法における各行 3 トークン開発履歴フォーマットの導入と評価

指導教員
肥後 芳樹 教授

報告者
近藤 偉成

令和 7 年 2 月 6 日

大阪大学 基礎工学部 情報科学科

令和 6 年度 特別研究報告

SZZ 手法における各行 3 トークン開発履歴フォーマットの導入と評価

近藤 偉成

内容梗概

SZZ 手法は，対象ソフトウェアの着目したバグについて，そのソフトウェアの開発履歴からそのバグを混入したコミットを自動で推測する方法である．SZZ 手法はソフトウェア工学の研究において広く利用されているが，特定の状況において推測が誤ってしまうことが課題である．この課題を解決するため，先行研究ではソースコードを各行が単一トークンで構成されるように整形した特殊な開発履歴を用いて SZZ 手法を適用する試みを行った．この特殊な開発履歴を使うことにより課題を解決できることがわかったが，また別の状況においてバグ混入コミットの推測が誤ってしまうことがわかった．そこで本研究では，通常の開発履歴を用いた場合や各行単一トークンの開発履歴を用いた場合にうまくバグ混入コミットを推測できない状況においても，正しくバグ混入コミットを推測できる開発履歴のフォーマットを提案する．具体的には，各行単一トークンの開発履歴に対して，各トークンに前後のトークンを加えた各行 3 トークンの開発履歴フォーマットである．実験として，68 の OSS プロジェクトを対象に，通常の開発履歴，各行単一トークンの開発履歴，本研究で提案するフォーマットの開発履歴を用いて SZZ 手法を適用し比較を行った．その結果，提案手法は誤った推測をするケースが増えるものの，通常の開発履歴と各行単一トークンの開発履歴の利点を両立することを示した．

主な用語

SZZ 手法
バグ修正コミット
バグ混入コミット

目 次

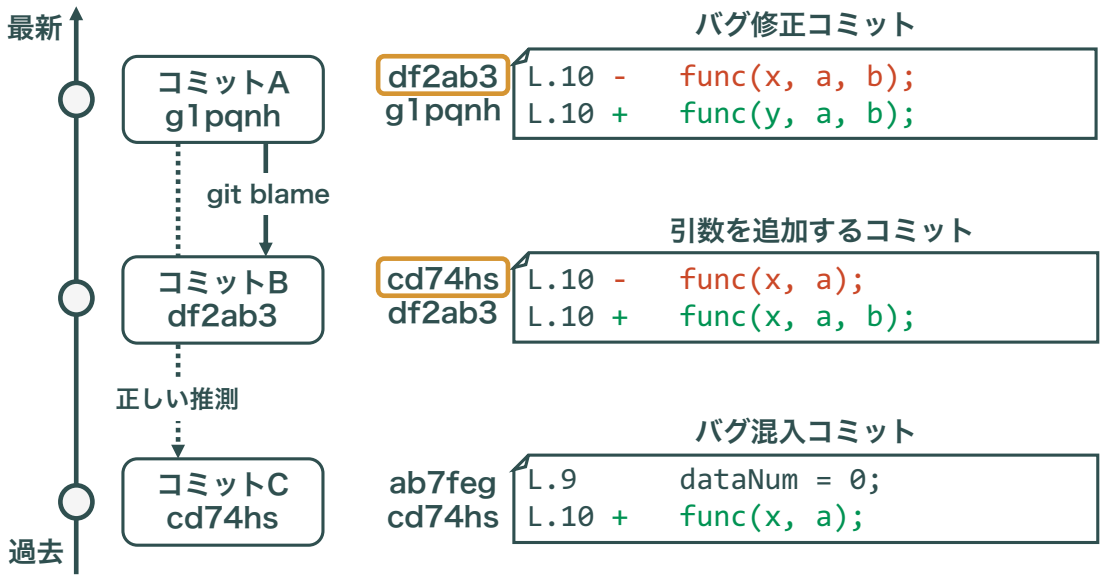

図 1: blame を利用して誤った推測をする例

1 まえがき

ソフトウェア開発は人間が手作業によって行うため，開発工程においてソフトウェアに誤り（ソフトウェアのバグ）が混入することは避けられない．ソフトウェア開発者はバグの混入を防ぐためにデバッグ作業を行うが，デバッグ作業は人的・金銭的に大きなコストがかかるため負担が大きい [1]．デバッグ作業のコスト削減のためにバグの分析，予測，修正などのバグに関する研究 [2, 3, 4, 5, 6, 7] が行われている．

バグに関する研究を行うために広く利用される手法のひとつに SZZ 手法 [8] がある．SZZ 手法はソフトウェアの開発履歴において，バグを混入したコミットを自動で推測する方法であり，バグに関する研究における分析対象のデータセット構築に利用される [4, 6, 9]．SZZ 手法はソースコードの版管理システムである Git の開発履歴と `blame` という機能 [10] を利用する．Git の開発履歴はコミットという単位で管理され，各コミットはソースコードの変更を行単位の削除および追加で管理する．`blame` は与えられたソースコードの行が追加されたコミットのハッシュ値を表示することができる．SZZ 手法では，バグを修正したコミットにおける変更行にバグが存在していたと仮定し，`blame` を利用して，その行が追加されたコミットをバグ混入コミットとして特定する．

SZZ 手法はバグを混入したコミットを推測するために広く利用されるが，その推測精度は高くないことが報告されている [11]．理由のひとつとして `blame` を利用したバグ混入コミット特定の限界がある [12]．図 1 に Git の開発履歴として 3 つのコミットを示す．コミット A

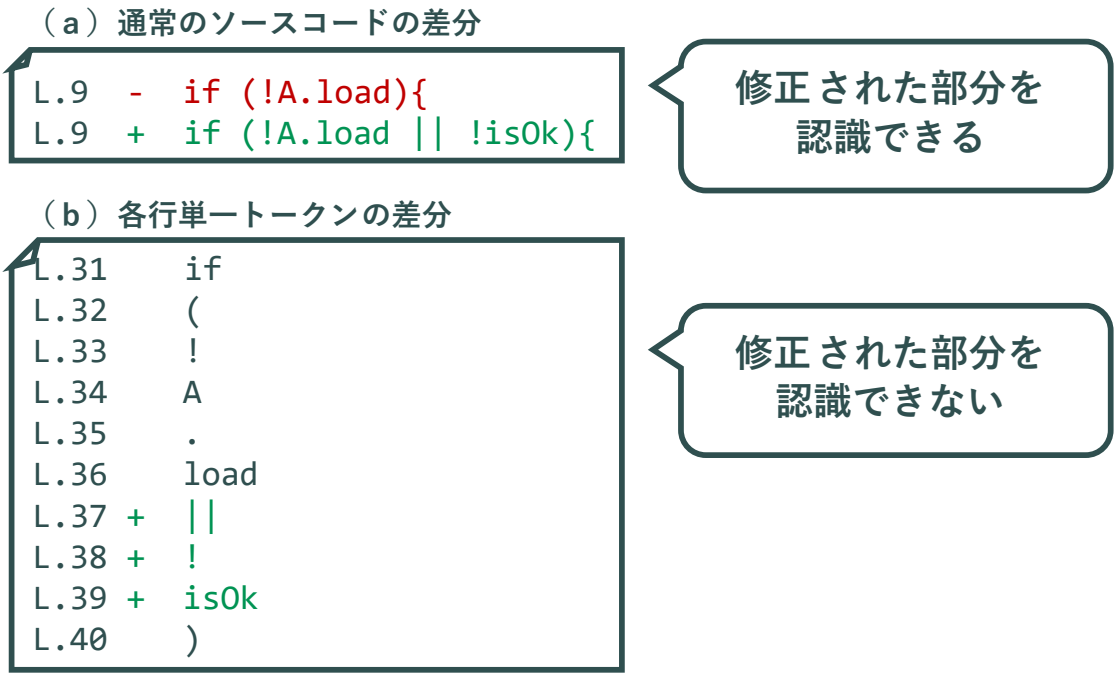

図 2: バグ修正コミットで各フォーマットで認識される差分

において 10 行目のバグの原因である変数 x が変数 y に修正されている．SZZ 手法は 10 行目にバグが混入していたと仮定し，`blame` を利用して 10 行目が追加されたコミット B を特定する．しかし，コミット B では 10 行目に対して引数を追加する変更しかしておらず，実際のバグ混入は引数 x を追加したコミット C で発生している．通常の Git の開発履歴は行単位で変更を管理するため，引数の追加などバグとは関係しない変更で，バグ混入コミットの特定に失敗し，SZZ 手法の推測精度の低下につながる．

SZZ 手法の推測精度向上のために，Git の開発履歴を各行が単一トークンで構成される開発履歴に整形する手法が提案されている [13]．単一トークンごとに `blame` を適用できるようになり，コメント変更のようなバグの混入とは関係がない変更の影響を無視できるようになる．しかし，各行単一トークンの開発履歴を利用することで新たな問題が起こることも報告されている [13]．例えば，図 2 のようにバグ修正コミットの変更が，各行単一トークンでは追加のみで表現される場合である．`blame` は与えられた行が追加されたコミットのハッシュ値を表示するため，追加行に対して適用してもその行を追加してバグ修正を行ったコミットを指してしまう．

そこで，本研究では，通常の開発履歴および各行単一トークンの開発履歴のどちらにおいても SZZ 手法の推測がうまくいかない状況において，正しくバグ混入コミットを推測できる新たな開発履歴フォーマットを提案する．具体的には，各行単一トークンの開発履歴に対

して，各トークンに前後のコンテキストを加える．各トークンに前後のコンテキストを加えることで，バグ修正コミットにおいて新たなトークンの追加によって修正が行われた場合でも，修正された箇所を明確に特定できるようになる．

本研究では，68 のオープンソースソフトウェア (OSS) プロジェクトを対象に各行 3 トークン開発履歴が SZZ 手法に与える影響を調査する．具体的には，通常の開発履歴，各行単一トークンの開発履歴，本研究で提案するフォーマットの開発履歴を用いて SZZ 手法を適用し，結果を比較することで以下の 3 つ研究設問（RQ）に対する回答を得ることを目指す．

RQ1：各行 3 トークンの開発履歴は SZZ 手法の精度を向上させるのか？

RQ2：各開発履歴で正しく推測するバグ混入コミットの特徴はどう異なるか？

RQ3：各開発履歴で誤って推測するバグ混入コミットの特徴はどう異なるか？

実験の結果，各行 3 トークン開発履歴を利用すると，正しく推測するバグ混入コミットの数を増加させることがわかった．そして，この開発履歴を利用することの利点と欠点を他の開発履歴との比較によって明らかにした．

以降，2 章では本研究の背景を先行研究の観点から述べる．3 章では本研究の提案手法を述べる．4 章では対象のデータセットと実験方法を述べる．5 章では RQ に対する実験結果，分析結果について述べる．6 章では実験結果に対する考察と妥当性への脅威を述べる．7 章では結論を述べる．

2　背景

本章では，本研究の背景として SZZ 手法について説明し，SZZ 手法が抱える課題やそれに対する先行研究について述べる．そして，先行研究の課題への解決策を述べる．

2.1　SZZ 手法

2.1.1　SZZ 手法の概要

SZZ 手法 [8] は，開発履歴を持つプロジェクトからバグを混入するコミットを特定するために広く利用されている手法である．具体的には以下の 2 つの手順を持つ．

手順（1）： バグ修正コミットの識別

手順（2）： バグ混入コミットの推定

手順（1）ではバグの修正を示唆するコミットメッセージを探すことでバグ修正コミットを識別する．そして，バグ修正コミットで変更された行を明らかにする．手順（2）では手順（1）で明らかになった変更行を追加した過去のコミットを `blame` を使って取り出す．取り出されたコミットをバグ混入コミットであるとする．

2.1.2　SZZ 手法の課題

SZZ 手法は広くソフトウェア工学研究において使われている [5, 6, 9] が，以下の 2 つの課題がありバグ混入コミット推定精度に悪影響を及ぼすことが知られている [11, 12, 13, 14, 15, 16, 17, 18].

課題（A）： バグが混入された行が変更されるコミットが複数回あると `blame` が誤った結果を返す．具体的には，コード整形やリファクタリング，コメントの変更などのバグに関係ない部分の変更である．

課題（B）： 行の追加によってバグの修正がされた場合，`blame` を適用する箇所が特定できず，バグ混入コミットを特定することができない．

これらの課題を解決することが求められる．

2.2　先行研究と課題

SZZ 手法の課題を解決することを目指して研究が行われている [11, 12, 13, 14, 15, 16, 17, 18]．Watanabe ら [13] は，課題（A）に対処するため，各行単一トークンの開発履歴を活用

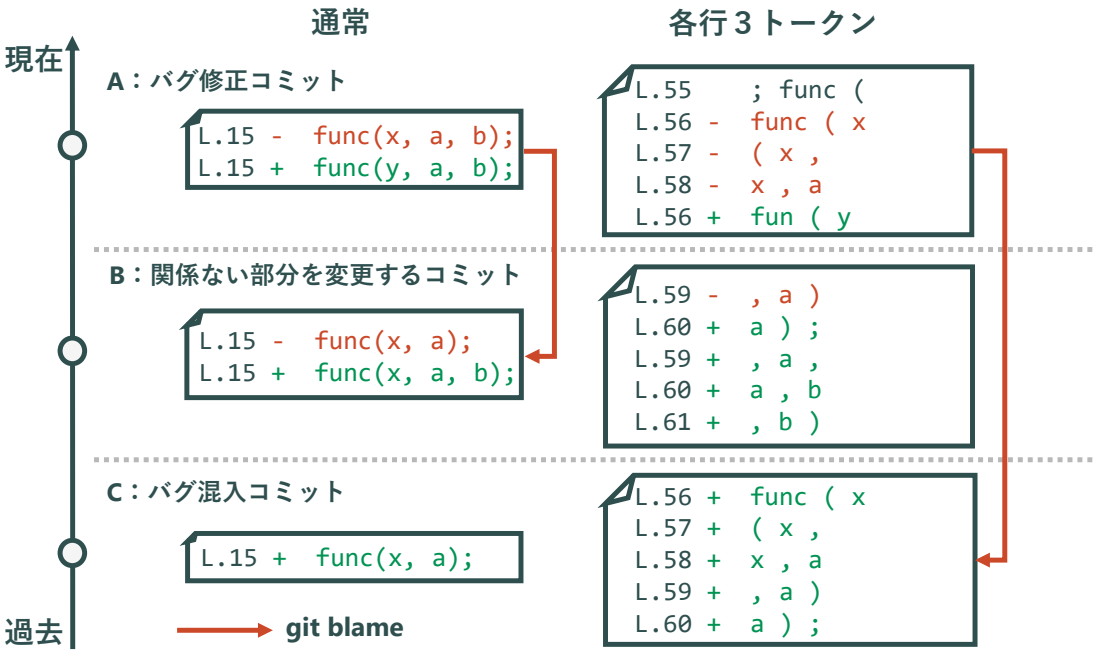

図 3: 課題（A）の状況における各行 3 トークンの blame

した SZZ 手法を提案した．通常の SZZ 手法は，行単位のコミットによって構成される Git の開発履歴を利用して blame によるバグ混入コミットの推定を行う．この手法では，Git の開発履歴を各行単一トークンのコミットによって構成される開発履歴に変換した上で blame を適用する．各行単一トークンの開発履歴は，トークン単位で変更を追跡できるため，図 1 のような例に対処できる．

しかし，各行単一トークンの開発履歴を使う場合に，通常の開発履歴では発生していなかった課題（B）が発生する場合がある．図 2（a）のコミット変更を各行単一トークンのコミットに変換すると図 2（b）になる．通常のコミットでは行の削除と追加による修正として Git が認識するが，各行単一トークンのコミットでは追加のみの修正として Git が認識する．そのため，課題（B）が発生しバグ混入コミットを推定できない．

2.3 解決策

図 1 及び図 2 の課題は各行単一トークンの開発履歴に対して各トークンに前後のコンテキストを加えることで解決できる．具体的には，各行単一トークンの開発履歴を各行 3 トークンの開発履歴に変換する．図 3 に，課題（A）の状況における通常の開発履歴と各行 3 トークンの開発履歴のコミットを示す．通常の開発履歴では，バグ混入コミットとしてバグの原因である変数 x とは関係ない部分を変更しているコミット B を特定してしまう．各行 3 トー

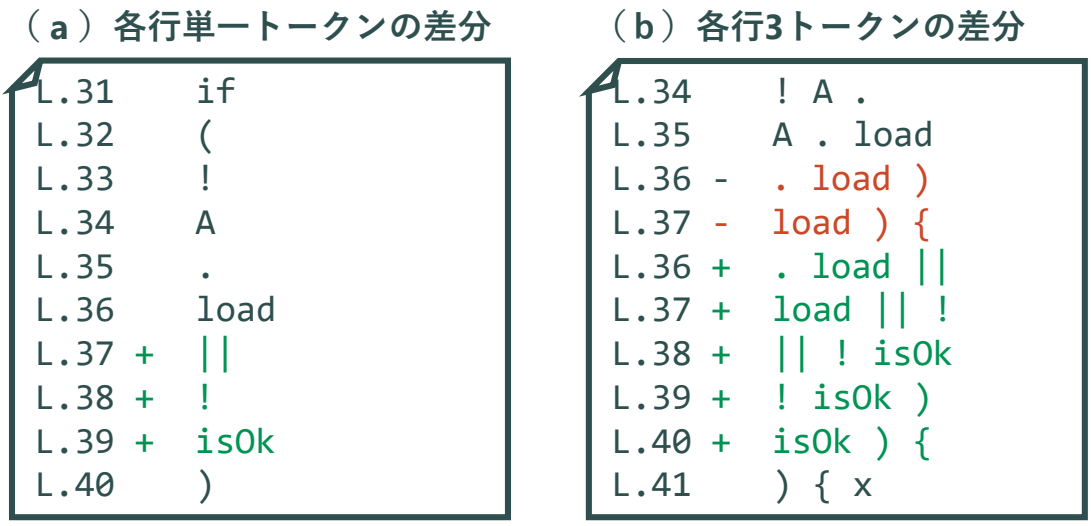

図 4: 課題（B）の状況における各行 3 トークンの差分

クンの開発履歴は行単位よりも細かく変更を追跡することができる．そのため，変数 x を正しく追跡し変数 x を混入したコミット C をバグ混入コミットとして正しく特定できる．

図 4 は開発履歴を各行 3 トークンに変換した際のコミットの変更である．各行単一トークンでは追加のみの変更になってしまうが，各行 3 トークンとすることで削除と追加を含んだ変更となる．これにより，課題（B）に対応できる．

このように，各トークンの前後のコンテキストを考慮した開発履歴を活用することで SZZ 手法の課題を解決できる．本研究では，各行 3 トークンの開発履歴を活用した SZZ 手法を既存の SZZ 手法と比較することで評価する．

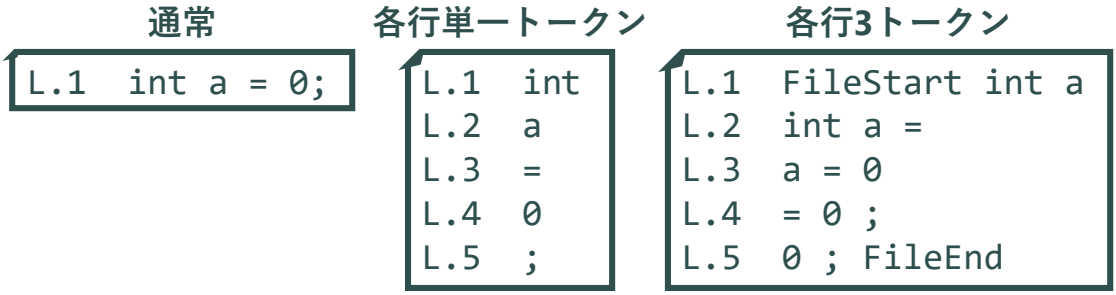

図 5: 各フォーマットのソースコードファイル

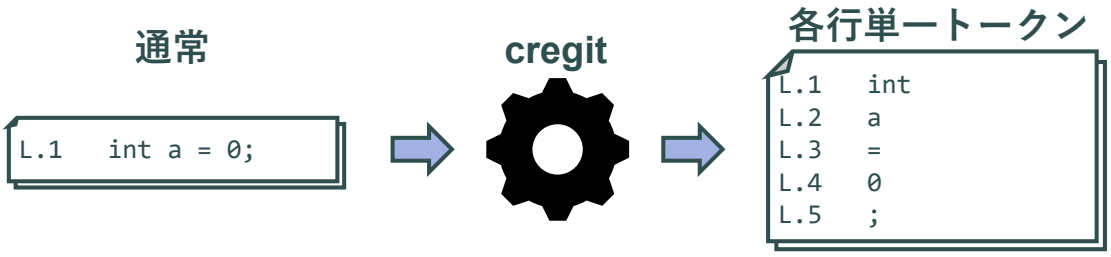

図 6: cregit による各行単一トークンへの変換

3 提案手法

本章では本研究で提案する各行 3 トークンの開発履歴の概要と作成方法，それを利用した SZZ 手法について述べる．

3.1 各行 3 トークンの開発履歴の概要

本研究では，対象プロジェクトの開発履歴から，各行 3 トークンに変換した開発履歴を作成し，各行 3 トークンの開発履歴に SZZ 手法を実行する手法を提案する．各行 3 トークンの開発履歴は図 5 のようになる．各行には 3 トークンが含まれており，各トークンは空白によって区切られる．例えば，図 5 の各行単一トークンのコードの 2 行目と各行 3 トークンのコードの 2 行目を比較すると，各行単一トークンでは変数 a のみであるのに対し，各行 3 トークンでは変数 a に前後のトークンである int と=が追加されている．なお，ソースコードファイルの最初のトークンおよび最後のトークンは，前もしくは後のトークンが存在しない．そのため，最初のトークンの前には FileStart，最後のトークンの後には FileEnd という文字列を追加することで 3 トークンとしている．

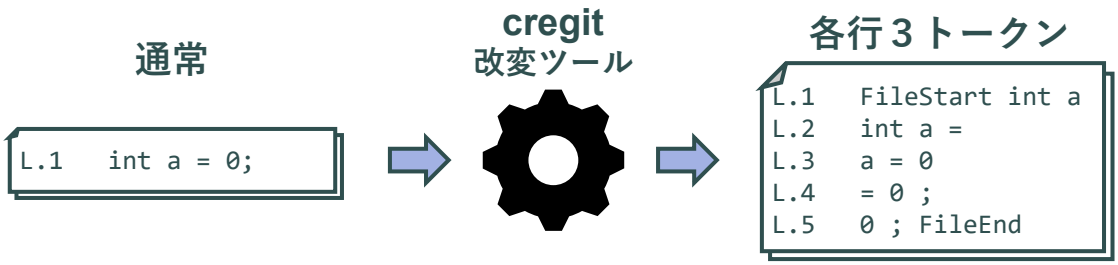

図 7: cregit 改変ツールによる各行 3 トークンへの変換

3.2 各行 3 トークンの開発履歴の作成方法

各行 3 トークンの開発履歴の作成には，先行研究 [13] で各行単一トークンの開発履歴を作成するために使用されていた cregit [19] というツールに変更を加えたものを使用した．cregit はソースコードファイルを解析し，ソースコードに含まれるすべてのトークンを含むファイルを生成する．生成されるファイルの各行は，各トークンとその種類であり，各行の順序は元のソースコードファイルにあらわれるトークンの順序と一致している．図 6 のように通常のファイルを cregit に与えると，各行単一トークンのファイルが生成される．なお，本論文ではわかりやすさを考慮し，各行単一トークンのソースコードにはトークンタイプを記載していない．本研究では，cregit の変換先のファイルへ書き込む処理の前に，各トークンに対して前後のトークンを追加する処理を行うよう変更し，各行 3 トークンの開発履歴を作成できるように cregit を改変した．図 7 の通常のファイルを cregit を改変したツールに与えると，各行 3 トークンのファイルが生成される．

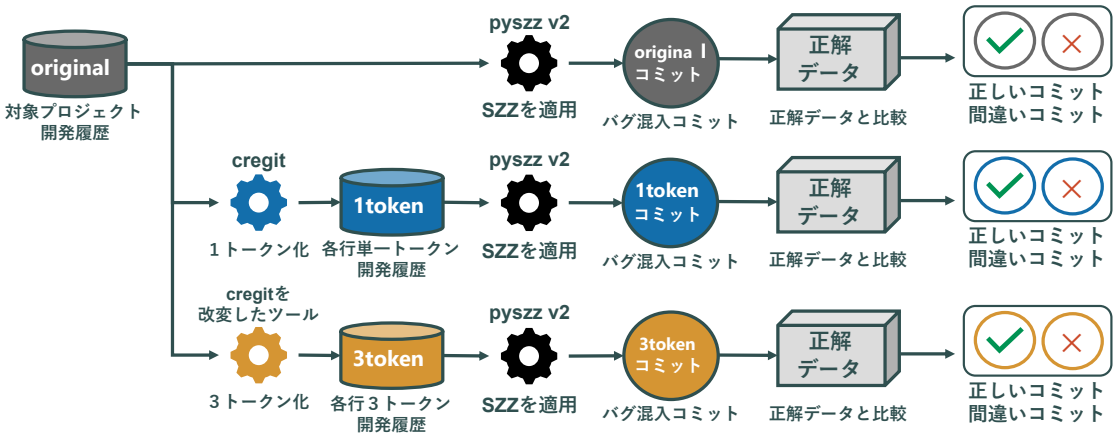

図 8: 実験の流れ

4 実験設定

本章では，実験設定として，実験対象のデータセット，実験の流れ，評価方法について述べる．

4.1 データセット

バグ混入コミットの正解データとして，開発者情報付きオラクルデータセット [20] を使用した．このデータセットは，バグ混入コミットのコミットハッシュを含むコミットメッセージを収集して作成された．開発者のコミットメッセージを利用してバグ修正コミットとバグ混入コミットのペアが手動で作成されており，品質が高い．このデータセットには，8 つの主要なプログラミング言語で書かれた合計 1,854 のリポジトリが含まれている．この実験では，主に Java で開発された 68 の OSS プロジェクトを対象とした．Java で開発されたプロジェクトからなるデータセットは，多くの先行研究で SZZ の変種の評価に利用されている．本実験のデータセットは，プロジェクト数は 68，合計コミット数は 997,287，バグありコミット数は 76 である．

表 1: 対象のデータセット

プロジェクト	コミット	バグありコミット
68	997,287	76

4.2 実験の流れ

図 8 に実験の流れを示す．まず，SZZ 手法に対する影響の比較を行う 3 つのフォーマットの開発履歴を作成した．3 つのフォーマットの開発履歴とは行単位の開発履歴，各行単一トークンの開発履歴，そして提案手法である各行 3 トークンの開発履歴である．各開発履歴の作成方法を示す．

行単位の開発履歴 対象とする 68 の OSS プロジェクトをクローンした通常の開発履歴である．

各行単一トークンの開発履歴 対象とする 68 の OSS プロジェクトの開発履歴に対して cregit を使用して作成する．cregit を使用して，各行が単一トークンで構成されたフォーマットの開発履歴に変換する．

各行 3 トークンの開発履歴 対象とする 68 の OSS プロジェクトの開発履歴に対して cregit に変更を加えたツールを使用して作成する．このツールを使用して，各行単一トークンの開発履歴に対して，各トークンに前後のトークンを追加した各行が 3 トークンで構成されたフォーマットの開発履歴に変換する．

次に，PySZZ v2 [20] というツールを用いて 3 つの開発履歴に対して SZZ 手法を適用する．PySZZ v2 は，入力として Git の開発履歴とバグ修正コミットのハッシュ値を受け取り，バグ混入コミットとして推測したコミットのハッシュ値を返す．このハッシュ値を開発者がラベル付けしたバグ混入コミットのハッシュ値と比較し，正しく推測したバグ混入コミットと誤って推測したバグ混入コミットに分類する．

4.3 評価

本実験では 2 つの観点から各行 3 トークンの開発履歴を活用した SZZ 手法を評価する．

バグ混入コミットの推定精度の評価

各開発履歴で，SZZ 手法によって推定されたバグ混入コミットの推定精度を評価する．推定精度の評価には，Precision，Recall，F1 score を用いる．これらの評価指標をもとに RQ1 に対する回答を得る．Precision はバグ混入コミットとして推測したコミットのうち，実際にバグ混入コミットであるコミットの割合を示す．Recall は実際にバグ混入コミットであるコミットのうち，バグ混入コミットとして推測したコミットの割合を示す．F1 score は Precision と Recall の調和平均であり，Precision と Recall の両方を考慮した指標である．各指標の算出方法は以下の通りである．

$$\text{Precision} = \frac{TP}{TP + FP} \tag{1}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

$$\text{F1 score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

ここで，TP（True Positive）は正しく推測したバグ混入コミットの数，FP（False Positive）はバグ混入コミットと判定したが実際はバグ混入コミットではなかったコミット数，FN（False Negative）はバグ混入コミットと判定しなかったが実際はバグ混入コミットであったコミット数である．

推測されるバグ混入コミットの特徴分析

各開発履歴を活用して正しく推測できたバグ混入コミットと誤ってバグ混入コミットとして特定されたコミットの特徴を目視分析する．目視でこれらのコミットのソースコードに対する変更を分析することで，開発履歴の違いが及ぼす影響を明らかにする．正しく推測できたバグ混入コミットの分析結果をもとに RQ2 に対する回答を得る．誤ってバグ混入コミットとして特定されたコミットの分析結果をもとに RQ3 に対する回答を得る．

5　実験結果

本章では実験結果について述べる．

5.1　RQ1: 各行 3 トークンの開発履歴は SZZ 手法の精度を向上させるのか？

発見 1：各行 3 トークンの開発履歴フォーマットは正しく推測できたバグ混入コミットの数を増加させる．

表 2 は各開発履歴フォーマットに対して SZZ 手法を適用した時のバグ混入コミットの推測結果の評価指標値である．各行 3 トークンの開発履歴を活用すると，76 件のバグのうち 53 件でバグ混入コミットを正しく推測できた．これは，行単位の開発履歴よりも 2 件多く，各行単一トークンの開発履歴よりも 10 件多い．各行 3 トークンの開発履歴は，3 つの開発履歴フォーマットの中で最も多くのバグ混入コミットを正しく推測できた．

発見 2：各行 3 トークンの開発履歴フォーマットは誤って推測するバグ混入コミットの数を増加させる．

FP の件数は行単位の開発履歴より 39 件多く，各行単一トークンの開発履歴より 25 件多い 137 件であった．バグ混入コミットではないコミットに対してバグ混入コミットと推測してしまう件数が，3 つの開発履歴フォーマットの中で最も多い． 図 9 より，行単位の開発履歴と比較して Precision は 0.063 低下し，F1 score は 0.061 低下する．

回答：各行 3 トークンの開発履歴を活用することで，正しく推測できたバグ混入コミットの数を増加させることができるが，誤って推測されるバグ混入コミットの数も増えてしまう．そのため，各行 3 トークンの開発履歴フォーマットはバグ混入コミットの推測精度を低下させる．

表 2: 異なる開発履歴フォーマットごとのバグ混入コミット推測結果の評価指標値

フォーマット	TP	FP	Precision	Recall	F1 Score
通常	51	98	0.342	0.671	0.459
各行単一トークン	43	112	0.277	0.566	0.372
各行 3 トークン	**53**	137	0.279	0.694	0.398

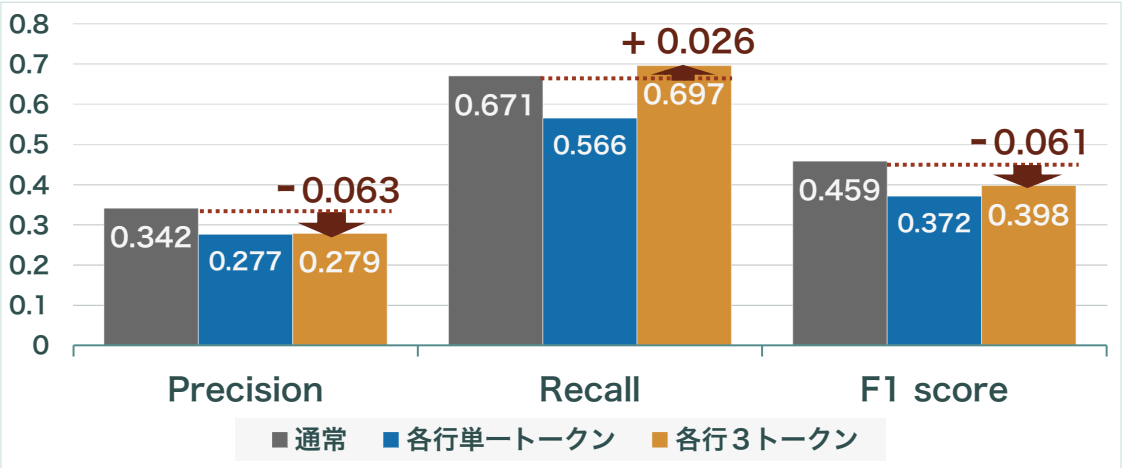

図 9: 各開発履歴のバグ混入コミット推定精度

5.2 RQ2: 各開発履歴で正しく推測するバグ混入コミットの特徴はどう異なるか？

発見 3：各行 3 トークンのフォーマットを使うことで，課題（B）によって見逃していたバグ混入コミットを発見可能になる．

図 10 は行単位の開発履歴（Line，赤色），各行単一トークンの開発履歴（Token，青色），および，各行 3 トークンの開発履歴（3Token，緑色）それぞれの TP の包含関係を図示している．各行 3 トークンの開発履歴でのみで発見可能であったバグ混入コミットは 5 件であった．これらは全て課題（B）によって発見されていなかった．つまり，行単位の開発履歴では追加のみで修正されているバグ修正コミットに対応するバグ混入コミットであり，`blame` が失敗するコミットである．図 11 は対象バグ混入コミットのバグを修正したコミットの差分の一部である．行単位の開発履歴のコミットでは追加のみの修正であるが，各行 3 トークンの開発履歴のコミットでは削除と追加による修正となる．そのため，各行 3 トークンの開発履歴では削除行に `blame` を適用してバグ混入コミットを特定できる．

発見 4：各行 3 トークンのフォーマットを使うことで，各行単一トークンの開発履歴で新たに発生する課題（B）を回避しバグ混入コミットを発見できる．

行単位の開発履歴および各行 3 トークンでのみ発見可能であったバグ混入コミットは 6 件であった．これらは全て各行単一トークンの開発履歴においてトークンの追加のみとなるバグ修正コミットに対応するバグ混入コミットである．つまり，各行単一トークンの開発履歴にすることで新たに課題（B）が発生していたことを示す．

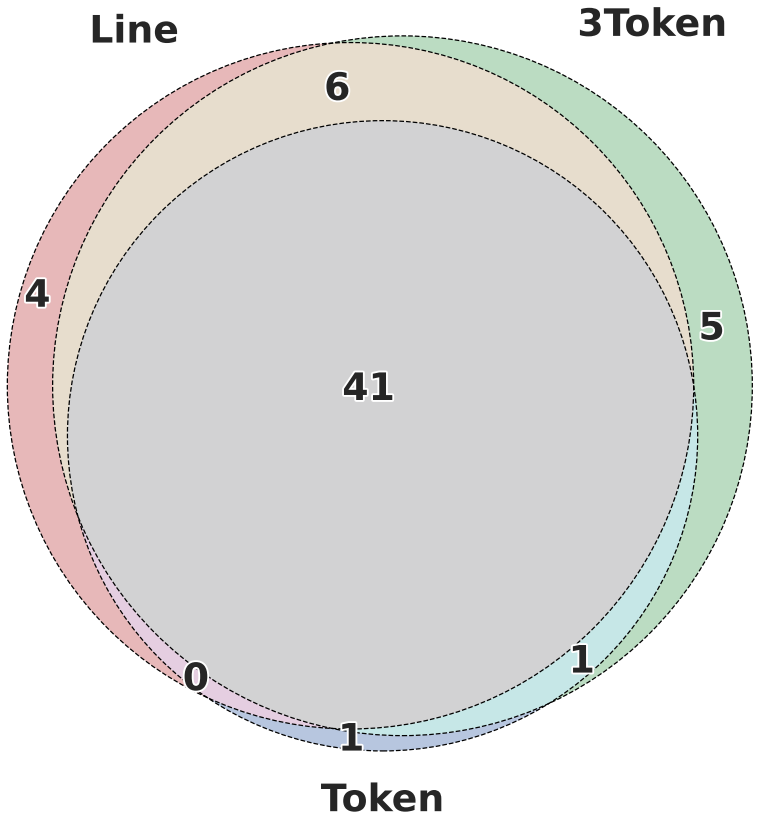

図 10: 各変更履歴の TP の包含関係

発見 5：各行 3 トークンのフォーマットを使うことで，Git の差分の取り方に影響を与え，通常の開発履歴では見逃すバグ混入コミットを発見可能になる場合と，見逃す場合が存在する．

各行単一トークンおよび各行 3 トークンの開発履歴でのみ発見可能であったバグ混入コミットは 1 件であった．図 12 は対象バグ混入コミットのバグを修正したコミットの差分の一部である．行単位の開発履歴のコミットでは 4 行目を削除し，新たなソースコードを 1 行目に追加する形で Git が差分が取られる．一方，各行 3 トークンでは関数呼び出しにおける引数を書き換える形で差分が取られる．この差分の取り方の違いにより，各行単一トークンおよび各行 3 トークンでのみでこのコミットに対応するバグ混入コミットが発見できた．これは，各行単一トークンと各行 3 トークンの開発履歴は細かく変更を追跡することができるため，行単位の開発履歴では見逃してしまうバグ混入コミットを発見できることを示している．一方で，各行単一トークンでのみ発見されたバグ混入コミット 1 件も同様に差分の取り方によって発見可能になった．そのため，差分の取り方の違いによって良くなる場合と悪く

```
 9       if(a == 0){
10         b = 0;
11 +       break;
12       }
13       x = 0;
```

(a) 行単位の開発履歴における差分

```
35       b       =       0
36       =       0       ;
37 -     0       ;       }
37 +     0       ;       break
38 +     ;       break   ;
39 +     break   ;       }
40       ;       }       x
```

(b) 各行 3 トークンの開発履歴における差分

図 11: 行が追加されて各行 3 トークンの差分が削除を含む例

```
1 +     func(a, c, null);
2       func(a, d, null);
3       func(b, c, null);
4 -     func(b, d, null);
```

(a) 行単位の開発履歴における差分

```
2       func (      a
3       (     a       ,
4 -     a     ,       d
5 -     ,     d       )
6 -     d     ,       p
4 +     a     ,       c
5 +     ,     c       )
6 +     c     ,       null
7       ,     null        )
```

(b) 各行 3 トークンの開発履歴における差分

図 12: 行単位と各行 3 トークンで差分の取り方が異なる例

なる場合の双方が観察された.

発見 6：各行 3 トークンでは，コード整形に関する変更を捉えられず推定できないバグ混入コミットがある.

行単位の開発履歴でのみ発見可能であったバグ混入コミットは 4 件である．このうち 2 件は if 文の追加によってバグ修正が行われている．この時，if 文追加されたことによって，インデントが追加される．このインデントが追加されたコードを `blame` によって追跡することでバグ混入コミットが見つかる．各行 3 トークンの開発履歴ではインデントの変更は追跡できず，このバグ混入コミットを見逃してしまう．残りの 2 件は，バグ修正コミットで削除された空行に対して `blame` を適用することで発見されるバグ混入コミットである．このようにインデントや空行などコード整形に関する変更に関連するバグ混入コミットを各行 3 トークンでは見逃してしまう.

回答：各行 3 トークンの開発履歴を活用することで，課題（B）の，行やトークンの追加によってバグが修正された場合でもバグ混入コミットを発見できる．これは，行単位や各行単一トークンの開発履歴では見逃してしまうバグ混入コミットを発見できることを示している．また，各行単一トークンの開発履歴と同様に細かい変更を追跡できるため，行単位の開発履歴では見逃してしまうバグ混入コミットを発見できる．しかし，コード整形に関する変更を追跡できず，バグ混入コミットを見逃してしまう場合がある．

5.3 RQ3: 各開発履歴で誤って推測するバグ混入コミットの特徴はどう異なるか？

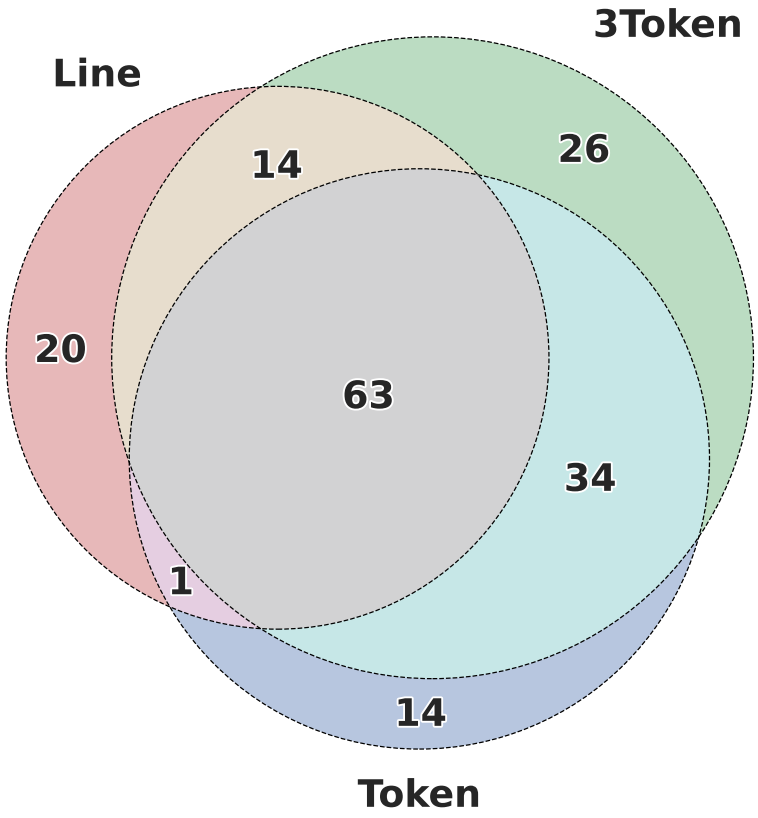

図 13: 各変更履歴の FP の包含関係

発見 7：各行 3 トークンでは `blame` の追跡対象が増えることにより誤ってバグ混入コミットと推定するコミット数が増加する．

図 13 は行単位の開発履歴（Line，赤色），各行単一トークン（Token，青色），および，

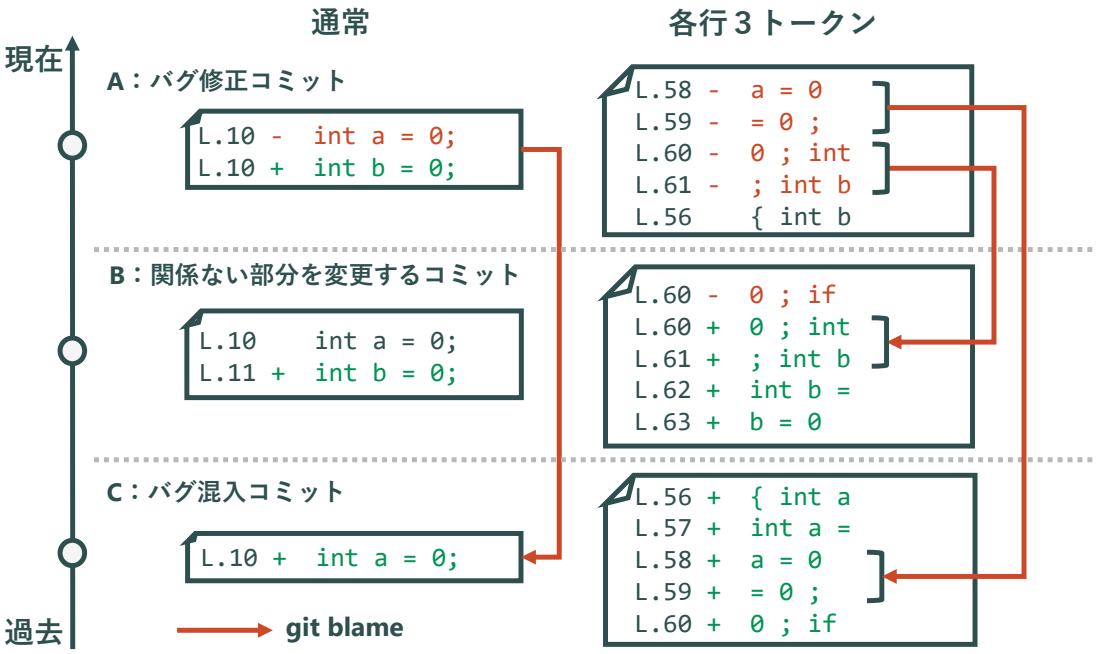

図 14: 各行 3 トークン開発履歴で誤った特定をする例

各行 3 トークン（3Token，緑色）それぞれの FP の包含関係図示している．各行 3 トークンでは 26 件の新たな FP を発生させている．図 14 は左側に行単位の開発履歴，右側に各行 3 トークンに変換した開発履歴を載せている．A，B，および，C の 3 つのコミットが載っており，A がバグ修正コミット，C がバグ混入コミットである．行単位の開発履歴の場合，A の修正行から C のみをバグ混入コミットとして特定可能である．一方で，各行 3 トークンとした場合，C を誤ってバグ混入コミットとして特定してしまう．なお，発見 3 および発見 4 からわかる通り，この影響により正しく推定可能なバグ混入コミットも存在する．発見 3 および 4 と発見 7 はトレードオフの関係にあると考えられる．

発見 8：行単位の開発履歴では空白や改行などのコード整形に関する変更を追跡したことが原因で誤ったバグ混入コミットの推定を行うケースが存在する．

各行単一トークンや各行 3 トークンの開発履歴を利用すると，改行や空白が原因の誤った推測を防ぐことが可能である．発見 6 とトレードオフの関係にある発見である．

発見 9：各行単一トークンの開発履歴では括弧（例:(),{}）やセミコロンなどの特定のトークンを追跡した結果，誤ったバグ混入コミットの推定を行うケースが存在する．

括弧やセミコロンなどのトークンを変更するコミットは少ない．そのため，括弧やセミコ

ロンはバグ混入コミットよりも前のコミットで最後に変更されている場合が多く，blame を適用すると誤ったバグ混入コミットを推測してしまう．

回答：各行 3 トークンの開発履歴を活用することで，blame の追跡対象が増えることにより誤ってバグ混入コミットと推定するコミット数が増加する．行単位の開発履歴では空白や改行などのコード整形に関する変更を追跡したことが原因で誤ったバグ混入コミットの推定を行うケースが存在する．各行単一トークンの開発履歴では括弧やセミコロンなどの特定のトークンを追跡した結果，誤ったバグ混入コミットの推定を行うケースが存在する．

6　考察

本章では実験結果の考察，今後の展望および妥当性への脅威について述べる．

6.1　各行 3 トークンが SZZ 手法に与える影響と今後の展望

本実験の発見は各行 3 トークンの開発履歴が SZZ 手法の精度向上に与える影響が限定的であることを示している．発見 1，3，4，5 のように精度向上に寄与するケースと，発見 2，5，6，7 に示すような精度を低下させるケースが観察された．精度を低下させるケースに対応する手法の研究が今後求められる．本実験の発見から考えられる今後の研究方針として以下の 2 つが挙げられる．

6.1.1　推測されるバグ混入コミットのフィルタリング

各行 3 トークンの開発履歴を活用することで，行単位の開発履歴よりも多くのバグ混入コミットを推測できることが示された．しかし，誤って推測されるバグ混入コミットの数も増加してしまう．この問題に対処するために，推測されたバグ混入コミットをフィルタリングする手法の提案が考えられる．例えば，発見 9 より，括弧やセミコロンなどの特定のトークンを追跡したことで推測されるバグ混入コミットはしばしば誤りであることがわかっている．この傾向を利用したフィルタリングとして，各行単一トークンの開発履歴を活用した SZZ 手法で推測されたバグ混入コミットから，”}”と”;”の追跡によって推測されたコミットの除去が考えられる．実際，除去を行うと各行単一トークンの開発履歴を活用した SZZ 手法の TP の数は 43 件で変化せず，FP の数は 112 件から 4 件減少し 108 件となった．同様のフィルタリングを各行 3 トークンの開発履歴を活用した SZZ 手法に応用することで FP の数を減らせる可能性がある．

6.1.2　周囲の行の変更に影響を受けない各行 3 トークンの開発履歴

発見 4，7 より，各行 3 トークンの開発履歴を活用することで，周囲の行の変更に影響を受けやすいことが示された．周囲の行の変更に影響を受けることで，行の追加による修正が行われた場合にバグ混入コミットを正しく推測できる場合があるが，誤って推測される場合もある．各行 3 トークンの開発履歴のみで誤って推測されるバグ混入コミットの数は 26 件と多く，その原因は周囲の行の変更に影響を受けやすいことが考えられる．この問題に対処する手法として，周囲の行の変更に影響を受けない各行 3 トークンの開発履歴フォーマットの作成が考えられる．具体的には，図 15 のように各行に含まれる 3 つのトークンは通常の開発履歴において全て同一の行に含まれるトークンとなる各行 3 トークンの開発履歴フォーマットである．図 15 では，通常の開発履歴における 30 行目は各行 3 トークンの開発履歴に

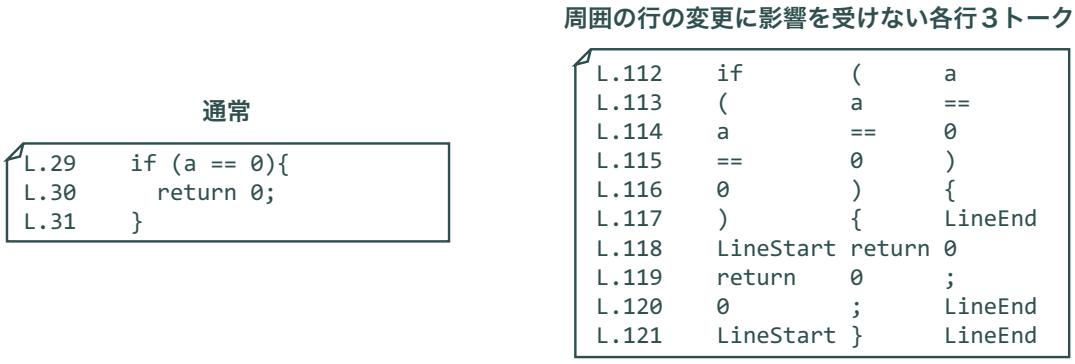

図 15: 周囲の行の変更に影響を受けない各行 3 トークンの開発履歴

おける 118，119，120 行目に対応している．このフォーマットを活用することで，FP の件数を大きく減少させる可能性がある．

6.2 妥当性への脅威

本研究の構成概念妥当性，外的妥当性，内的妥当性についてそれぞれ述べる．

6.2.1 構成概念妥当性

本研究では，各行 3 トークンの開発履歴フォーマットに SZZ 手法を適用する実験を行った．各行に含めるトークン数はパラメータであり，他のトークン数では異なる結果が得られる可能性がある．しかし，通常の開発履歴フォーマットおよび各行単一トークンのフォーマットの中間の粒度を用いた場合に結果が変化することを示した点は重要な示唆である．今後の研究として，各行に含めるトークン数を他の値に変化させた時の推定精度への影響分析や，別のコンテキスト考慮手段の提案などを行っていく必要がある．

実験に用いたデータセットの正解データの品質は本研究の妥当性に強く影響する．本研究で利用したバグ混入コミットの正解データは，開発者から提供された情報に依存している．正解データの正しさを目視で確認したところ，一部の正解データが正しくなかった．このような正解データの品質の問題は，実験結果に偏りを生じさせる可能性がある．今後，より多くのデータセットを活用することで妥当性向上を目指すことや，データセットの品質向上を目指したデータクリーニングなどを行う必要がある．

6.2.2 外的妥当性

本研究では，開発者情報付きオラクルデータセットに含まれる Java で書かれたプロジェクトを対象に実験を行った．そのため，本研究の結果は他のデータセットやプログラミング

言語には適用できない可能性がある．今後の研究では，対象のデータセットやプログラミング言語を追加し，本研究の結果の一般化可能性について検証する必要がある．

6.2.3 内的妥当性

本研究では，3 つの開発履歴を利用した SZZ 手法によって推測されたバグ混入コミットの特徴を目視で分析した．そのため，この分析には目視分析を行った著者の主観が混入している可能性がある．

7　まとめ

本研究では，SZZ 手法の精度向上を目指し，従来の行単位および各行単一トークンによる開発履歴の課題を克服するため，各行 3 トークンを基本単位とした新たな開発履歴フォーマットを提案した．そして，通常の開発履歴，各行単一トークンの開発履歴，各行 3 トークンの開発履歴が SZZ 手法に与える影響を比較した．その結果，各行 3 トークンの開発履歴が SZZ 手法に与える影響について以下の結果を得た．

- 各行 3 トークンの開発履歴を利用することで，行単位の開発履歴と各行単一トークンの開発履歴の両方の課題を解決し，正しく推測するバグ混入コミットの数を増加させる．
- 各行 3 トークンの開発履歴を利用することで，誤って推測されるバグ混入コミットの数が増加し，バグ混入コミットの推定精度を低下させる．

今後の研究課題として，精度を低下させるケースに対応するため，各行に含めるトークン数を変化させたときの推定精度の影響分析や，誤った推測を減らすためのフィルタリング手法の開発などが考えられる．

謝辞

大阪大学大学院情報科学研究科コンピュータサイエンス専攻肥後芳樹教授には，研究の着想から研究の進行にあたっての助言，論文執筆や発表資料の作成に至るまで多大なるご指導を賜りました．肥後芳樹教授の適切なご指導のもと，本研究を進めることができましたことを深く感謝いたします．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 Kula Raula Gaikovina 教授には，中間報告会等において，研究の方針に対する助言やご指摘を賜りました．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻松下誠准教授には，中間報告会等において，研究の方針に対する助言や，発表に対する的確なご指摘を賜りました．

九州大学システム情報科学研究院情報知能工学部門近藤将成助教には，共同研究を通じて研究の進行や論文執筆，発表資料の作成にあたって多大なるご指導を賜りました．近藤将成助教との議論やアイデアの共有を通じて，研究活動において多くのことを学ぶことができましたことを深く感謝いたします．

九州大学システム情報科学研究院情報知能工学部門亀井靖高教授には，共同研究の機会を与えていただき，本研究の進行にあたって貴重なご意見を賜りました．九州大学での研究活動を通じて，多くのことを学ぶことができましたことを深く感謝いたします．

ノートルダム清心女子大学情報デザイン学部情報デザイン学科神田哲也准教授には，中間報告会等において，研究の方針に対する助言や発表資料に対する的確なご指摘を賜りました．

The University of Victoria の Daniel M German 教授には，ツールの提供や研究の進行にあたって，貴重な助言を賜りました．

東京科学大学情報理工学院林晋平教授には，研究会におきまして，本研究における本質的な課題について貴重なご意見を賜りました．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 松本 真佑 助教には，研究会におきまして，本研究の進行にあたって貴重なご意見を賜りました．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 事務職員 軽部 瑞穂 氏は，私たち学生が快適な環境で研究を行うため，研究室の環境整備や事務的作業といったご支援を賜りました．

最後に，その他様々なご指導，ご助言を賜りました，大阪大学大学院情報科学研究科コンピュータサイエンス専攻肥後研究室と，九州大学システム情報科学研究院情報知能工学部門亀井研究室の皆様に深く感謝いたします．

参考文献

[1] Tom Britton, Lisa Jeng, Graham Carver, Paul Cheak, and Tomer Katzenellenbogen. Increasing software development productivity with reversible debugging. `https://undo.io/media/uploads/files/Undo_ReversibleDebugging_Whitepaper.pdf`.

[2] L. Tan, C. Liu, Z. Li, X. Wang, Y. Zhou, and C. Zhai. Bug characteristics in open source software. *Empirical Software Engineering*, Vol. 19, No. 6, pp. 1665–1705, 2014.

[3] C. Le Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer. A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each. In *Proceeding of the 34th International Conference on Software Engineering*, pp. 3–13, 2012.

[4] Claire Le Goues, Michael Pradel, Abhik Roychoudhury, and Satish Chandra. Automatic program repair. *IEEE Software*, Vol. 38, No. 04, pp. 22–27, 2021.

[5] Mario Luca Bernardi, Gerardo Canfora, Giuseppe A. Di Lucca, Massimiliano Di Penta, and Damiano Distante. Do developers introduce bugs when they do not communicate? the case of eclipse and mozilla. In *Proceeding of the 16th Europe Conference on Software Maintenance and Rewngineering*, pp. 139–148, 2012.

[6] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi. A large-scale empirical study of just-in-time quality assurance. *IEEE Transactions on Software Engineering*, Vol. 39, No. 6, pp. 757–773, 2013.

[7] F. Servant and J. A. Jones. Fuzzy fine-grained code-history analysis. In *Proceeding of the 39th International Conference on Software Engineering*, pp. 746–757, 2017.

[8] J. Sliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes. *ACM SIGSOFT Software Engineering Notes*, Vol. 30, No. 4, pp. 1–5, 2005.

[9] Rafael Michael Karampatsis and Charles Sutton. Manysstubs4j dataset (2.0). `https://doi.org/10.5281/zenodo.3653444`.

[10] S. Chacon and J. Long. Git - git-blame documentation. `https://git-scm.com/docs/git-blame`, 2024.

[11] Sunghun Kim, Thomas Zimmermann, Kai Pan, and E. James Jr. Whitehead. Automatic identification of bug-introducing changes. In *Proceeding of the 21st IEEE/ACM International Conference on Automated Software Engineering*, pp. 81–90, 2006.

[12] Christophe Rezk, Yasutaka Kamei, and Shane McIntosh. The ghost commit problem when identifying fix-inducing changes: An empirical study of apache projects. *IEEE Transactions on Software Engineering*, Vol. 48, No. 9, pp. 3297–3309, 2022.

[13] Hiroya Watanabe, Masanari Kondo, Eunjong Choi, and Osamu Mizuno. Benefits and pitfalls of token-level szz: An empirical study on oss projects. In *Proceeding of the IEEE International Conference on Software Analysis, Evolution and Reengineering*, pp. 776–786, 2024.

[14] Chadd Williams and Jaime Spacco. Szz revisited: verifying when changes induce fixes. In *Proceeding of the 2008 Workshop on Defects in Large Software Systems*, pp. 32–36, 2008.

[15] S. Davies, M. Roper, and M. Wood. Comparing text-based and dependence-based approaches for determining the origins of bugs. *Journal of Software: Evolution and Process*, Vol. 26, No. 9, pp. 117–139, 2014.

[16] T. Zimmermann, S. Kim, A. Zeller, and E. J. Whitehead. Mining version archives for co-changed lines. In *Proceeding of the 2006 International Workshop on Mining Software Repositories*, pp. 72–75, 2006.

[17] Edmilson Campos Neto, Daniel Alencar da Costa, and Uirá Kulesza. Revisiting and improving szz implementations. In *Proceeding of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 1–12, 2019.

[18] Lingxiao Tang, Lingfeng Bao, Xin Xia, and Zhongdong Huang. Neural szz algorithm. In *Proceeding of the 38th IEEE/ACM International Conference on Automated Software Engineering*, pp. 1024–1035, 2023.

[19] D. M. German, B. Adams, and K. Stewart. “ cregit: Token-level blame information in git version control repositories. *Empirical Software Engineering*, Vol. 24, No. 4, pp. 2725–2763, 2019.

[20] Giovanni Rosa, Luca Pascarella, Simone Scalabrino, Rosalia Tufano, Gabriele Bavota, Michele Lanza, and Rocco Oliveto. A comprehensive evaluation of szz variants through a developer-informed oracle. *Journal of Systems and Software*, Vol. 202, p. 111729, 2023.